# Coda

- Single location-transparent UNIX FS.
- Scalability - coarse granularity (whole-file caching, volume management)
- First class (server) replication and client caching (second class replication)
- Optimistic replication & consistency maintenance.

→ Designed for disconnected operation for mobile computing clients



# Disconnected and Weakly Connected Coda

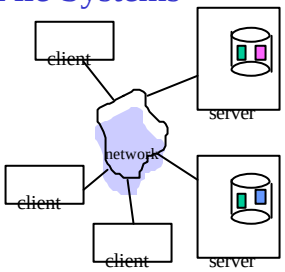
Satya, Kistler, Mummert, Ebling, Kumar, and Lu, "Experience with Disconnected Operation in a Mobile Computing Environment", *USENIX Symp. On Mobile and Location-Independent Computing*, 1993.

Mummert, Ebling, and Satya, "Exploiting Weak Connectivitiy for Mobile File Access", *SOSP95*.



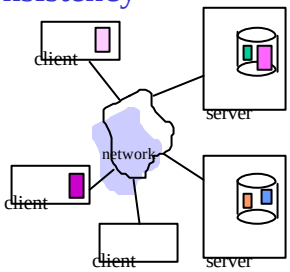
# Distributed File Systems

- Naming
  - Location transparency/ independence
- Caching
  - Consistency
- Replication
  - Availability and updates




# Cache Consistency

- Location of cache on client - disk or memory
- Update policy
  - write through
  - delayed writeback
  - write-on-close
- Consistency
  - Client does validity check, contacting server
  - Server call-backs



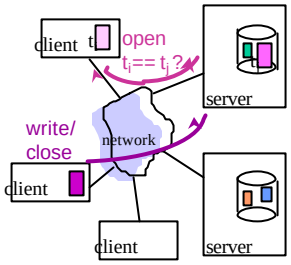
# File Cache Consistency

- Caching is a key technique in distributed systems.
  - The *cache consistency problem*: cached data may become *stale* if cached data is updated elsewhere in the network.
- Solutions:
  - *Timestamp invalidation* (NFS).
    - Timestamp each cache entry, and periodically query the server: "has this file changed since time *t*?"; invalidate cache if stale.
  - *Callback invalidation* (AFS).
    - Request notification (callback) from the server if the file changes; invalidate cache on callback.
  - *Leases* (NQ-NFS) [Gray&Cheriton89]



# Sun NFS Cache Consistency

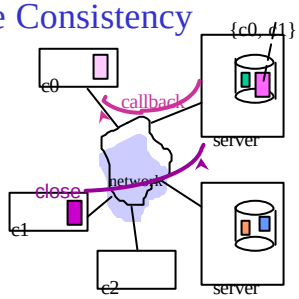
- Server is ***stateless***
- Requests are self-contained.
- ***Blocks*** are transferred and cached in memory.
- Timestamp of last known mod kept with cached file, compared with "true" timestamp at server on Open. (Good for an interval)
- Updates delayed but flushed before Close ends.






# AFS Cache Consistency

- Server keeps state of all clients holding copies (copy set)
- ***Callbacks*** when cached data are about to become stale
- Large units (whole files or 64K portions)
- Updates propagated upon close
- Cache on local disk –> memory






# NQ-NFS Leases

- In NQ-NFS, a client obtains a *lease* on the file that permits the client's desired read/write activity.
  - "A lease is a ticket permitting an activity; the lease is valid until some expiration time." - temporary statefulness
  - A *read-caching lease* allows the client to cache clean data.
    - **Guarantee**: no other client is modifying the file.
  - A *write-caching lease* allows the client to buffer modified data for the file. Must push data before expiration
    - **Guarantee**: no other client has the file cached.
- Leases may be revoked by the server if another client requests a conflicting operation (server sends *eviction notice*). Push in *write_slack* period.

## Explicit First-class Replication

- File name maps to set of replicas, one of which will be used to satisfy request
  - Goal: availability
- Update strategy
  - Atomic updates - all or none
  - Primary copy approach
  - Voting schemes
  - Optimistic, then detection of conflicts

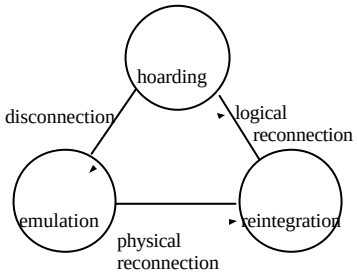
## Optimistic vs. Pessimistic

- High availability Conflicting updates are the potential problem - requiring detection and resolution.

- Avoids conflicts by holding of shared or exclusive locks.
- How to arrange when disconnection is involuntary?
- Leases [Gray, SOSP89] puts a time-bound on locks but what about expiration?



## Client-cache State Transitions






## Prefetching

- To avoid the access latency of moving the data in for that first cache miss.
- Prediction! "Guessing" what data will be needed in the future.
  - It's not for free:
    Consequences of guessing wrong
    Overhead

## Hoarding - Prefetching for Disconnected Information Access

- Caching for availability (not just latency)
- Cache misses, when operating disconnected, have no redeeming value.
  (Unlike in connected mode, they can't be used as the triggering mechanism for filling the cache.)
- How to preload the cache for subsequent disconnection? Planned or unplanned.
- What does it mean for replacement?



## Hoard Database

- Per-workstation, per-user set of pathnames with priority
- User can explicitly tailor HDB using scripts called *hoard profiles*
- Delimited observations of reference behavior (snapshot spying with bookends)



## Coda Hoarding State

- Balancing act - caching for 2 purposes at once:
  - performance of current accesses,
  - availability of future disconnected access.
- Prioritized algorithm -
  Priority of object for retention in cache is
  **f(hoard priority, recent usage).**
- Hoard walking (periodically or on request) maintains equilibrium - no uncached object has higher priority than any of cached objects

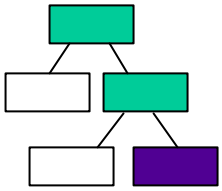


## The Hoard Walk

- Hoard walk - phase 1 - reevaluate name bindings (e.g., any new children created by other clients?)
- Hoard walk - phase 2 - recalculate priorities in cache and in HDB, evict and fetch to restore equilibrium

## Hierarchical Cache Mgt

- Ancestors of a cached object must be cached in order to resolve pathname.
- Directories with cached children are assigned infinite priority





## Callbacks During Hoarding

- Traditional callbacks - invalidate object and refetch on demand
- With threat of disconnection
  - Purge files and refetch on demand or hoard walk
  - Directories - mark as stale and fix on reference or hoard walk, available until then just in case.

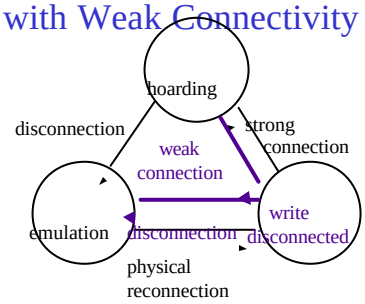


## Emulation State

- Pseudo-server, subject to validation upon reconnection
- Cache management by priority
  - modified objects assigned infinite priority
  - freeing up disk space - compression, replacement to floppy, backout updates
- Replay log also occupies non-volatile storage (RVM - recoverable virtual memory)



## Client-cache State Transitions with Weak Connectivity



## Cache Misses with Weak Connectivity

- At least now it's possible to service misses but $$$ and it's a foreground activity (noticable impact). Maybe **not**
- User patience threshold - estimated service time compared with what is acceptable
- Defer misses by adding to HDB and letting hoard walk deal with it
- User interaction during hoard walk.



## Other Hoarding Strategies

- Detection of "file working sets"
  Tait, Acharya, Lei, and Chang, "Intelligent File Hoarding for Mobile Computers", MOBICOM95.
- Capture semantic relationships among files in "semantic distance" measure - SEER
  Kuenning and Popek, "Automated Hoarding for Mobile Computers", SOSP97.



## Energy Implications?

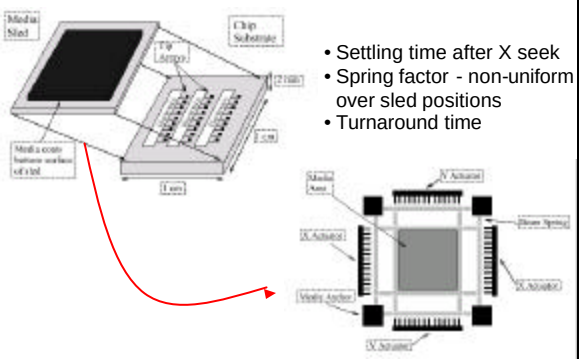
- Avoiding continuous wireless connectivity, on purpose, to save energy
- Using remote storage as primary repository or backup



## MEMS-based Storage

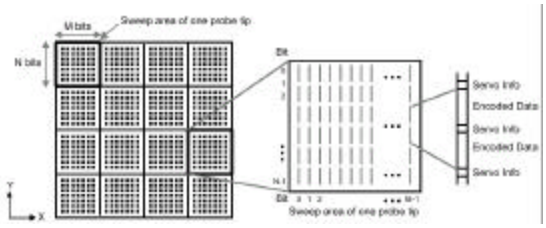
Griffin, Schlosser, Ganger, Nagle

- Paper in OSDI 2000 on OS Management
- Comparing MEMS-based storage with disks
  - Request scheduling
  - Data layout
  - Fault tolerance
  - Power management



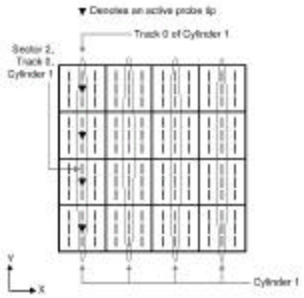

- Settling time after X seek
- Spring factor - non-uniform over sled positions
- Turnaround time

## Data on Media Sled






## Disk Analogy




- 16 tips
- MxN = 3 x 280
- Cylinder – same x offset
- 4 tracks of 1080 bits, 4 tips
- Each track – 12 sectors of 80 bits (8 encoded bytes)
- Logical blocks striped across 2 sectors



## Logical Blocks and LBN


- Sectors are smaller than disk
- Multiple sectors can be accessed concurrently
- Bidirectional access

## Comparison

MEMS

- Positioning – X and Y seek (0.2-0.8 ms)
- Settling time 0.2ms
- Seeks near edges take longer due to springs, turnarounds depend on direction – it isn't just distance to be moved.
- More parts to break
- Access parallelism

Disk

- Seek (1-15 ms) and rotational delay
- Settling time 0.5ms
- Seek times are relatively constant functions of distance
- Constant velocity rotation occurring regardless of accesses

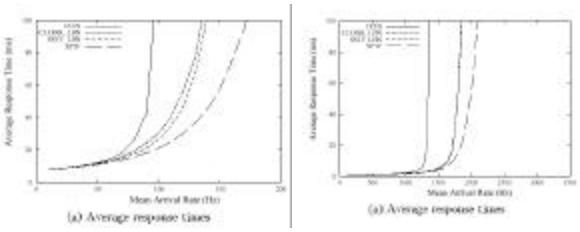


## Specific Parameters for Simulation Study

| | |
|---|---|
| device capacity | 3.2 GB |
| number of tips | 6400 |
| maximum concurrent tips | 1280 |
| sled acceleration | 803.6 m/s² |
| sled access speed | 28 mm/s |
| constant settling time | 0.22 ms |
| spring factor | 75% |
| per-tip data rate | 0.7 Mbit/s |
| media bit cell size | 40×40 nm |
| bits per tip region (M×N) | 2500×2440 |
| data encoding overhead | 2 bits per byte |
| servo overhead per 8 bytes | 10 bits (11%) |
| command processing overhead | 0.2 ms/request |
| on-board cache memory | 0 MB |
| external bus bandwidth | 100 MB/s |



## Request Scheduling



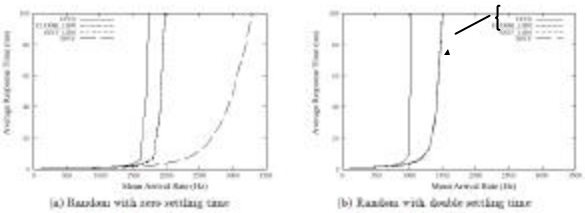
Quantum Atlas 10K Disk

MEMS

Random Workload



## Impact of Settling Time



Y matters – only captured by SPTF

X dominates

## Data Placement

- Offset from center matters to seek time – small data are placed in centermost subregions
- Positioning is relatively insignificant for large transfers – sequential streaming data placed in outer subregions
- Compared to organpipe policy– most *frequently* accessed data in middle disk tracks
- All better than nothing at all



## Failure and Power

- Error Correcting Code computed horizontally across tips (missing sector, bad tip) and vertically within sector (bad sector)
- Remap sector under spare tip allocated in each track
- Idle mode stops sled and powers down electronics
- Restart is fast 0.5ms and no power spike to "spin up." Immediate-idle (no timeout policy).



## Conclusions
(according to Ganger)

B-o-r-r-r-i-n-g from OS p.o.v.

– MEMS are simpler to manage